

Модуль 3.
ПРОЦЕДУРЫ И ФУНКЦИИ

СОДЕРЖАНИЕ

1. Процедуры и функции. Синтаксис, примеры использования	3
2. Глобальные и локальные переменные	7
3. Передача параметров по значению и по ссылке	9
3.1. Передача параметра по значению	9
3.2. Передача параметра по ссылке.....	14
3.3. Параметры по умолчанию	18
4. Рекурсия.....	20
5. Задания для самостоятельного выполнения	24
Литература.....	27

1. Процедуры и функции. Синтаксис, примеры использования

Очень часто при выполнении программ одни и те же действия осуществляются несколько раз над различными данными (например, нужно найти факториал нескольких чисел). В некоторых случаях встречаются просто повторения какого-либо участка кода (например, нужно вывести набор данных до и после выполнения каких-либо преобразований). В таких случаях было бы полезно выделить повторяющиеся фрагменты в отдельную программную единицу (подпрограмму) и использовать ее по мере необходимости.

Подпрограмма – это группа операторов, оформленная в виде самостоятельной программной единицы. Она записывается однократно, а в соответствующих местах программы обеспечивается лишь обращение к ней по имени – вызов.

Использование подпрограмм позволяет

- сократить объем программы,
- улучшить структуру программы с точки зрения ее читаемости и наглядности,
- уменьшить вероятность ошибок и облегчить процесс отладки.

Отметим, что не всегда подпрограммы используются для повторяющихся фрагментов программы. В некоторых случаях, особенно при решении сложных, больших задач, бывает удобно разбить исходную задачу на подзадачи, программируя и отлаживая каждый фрагмент по отдельности. Такой подход позволяет изменять, модифицировать каждую часть независимо от других.

Как правило, программа состоит из *основной программы* (с нее начинается работа) и некоторого количества подчиненных ей подпрограмм – отдельных функционально независимых частей программы. Вызов подпрограммы происходит следующим образом (см. рис. 1):

- 1) приостановка работы основной программы;
- 2) копирование параметров, создание локальных для подпрограммы переменных;
- 3) передача управления на первый оператор вызванной подпрограммы;
- 4) выполнение подпрограммы;
- 5) удаление локальных для подпрограммы переменных;
- 6) возвращение управления основной программе.

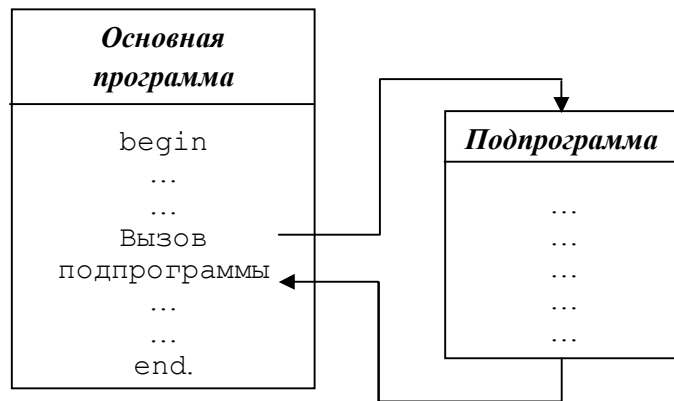


Рис. 1. Вызов подпрограммы

Основная программа может вызвать сколько угодно подпрограмм. Подпрограммы можно разделить на библиотечные и пользовательские. Примеры библиотечных подпрограмм: `sqr`, `sin`, `abs`, `write`, `read` и т. д. К пользовательским подпрограммам относятся подпрограммы, реализованные программистом в данной программе.

В языке Паскаль механизм подпрограмм реализуется в виде процедур и функций. Они различаются синтаксическим оформлением, назначением, способом вызова и передачей возвращаемого значения.

Процедура – это произвольное число операторов, снабженных именем, по которому и производится вызов процедуры. Любая процедура начинается с заголовка, который является ее обязательной частью (в отличие от заголовка программы). Он состоит из служебного слова `procedure`, за которым следует имя процедуры, а в круглых скобках – список формальных параметров; в конце ставится точка с запятой. После заголовка могут идти те же разделы, что и в программе (переменные, типы и т. п.). Таким образом, общий вид любой процедуры таков:

```
procedure имя (список_формальных_параметров);
раздел описаний
begin
    операторы – тело процедуры
end;
```

Функция предназначена для того, чтобы вычислять только одно значение. Часто функции используются как операнды в выражениях. Заголовок функции состоит из зарезервированного слова `function`, за которым идет имя функции, затем в круглых скобках идет список

формальных параметров, после чего через двоеточие записывается тип результата функции. Кроме того, в теле функции обязательно должен быть хотя бы один оператор присваивания, где в левой части стоит имя функции, а в правой – ее значение. Иначе значение функции не будет определено. В остальном, описание функций такое же, как и процедур.

Общий вид функции:

```
function имя (список_формальных_параметров) :
                                         тип_возвращаемого_значения;
раздел описаний
begin
    операторы
    имя := значение;
end;
```

Вместо имени функции внутри тела функции в операторе присваивания можно использовать специальную переменную с именем `result` (описывать ее в разделе описаний не нужно).

```
result := <значение>;
```

В отличие от имени функции переменную `result` можно использовать и в выражениях (если имя функции использовать в теле функции в выражении, то это будет расцениваться как рекурсивный вызов).

В разделе описаний подпрограммы, как и в разделе описаний основной программы, могут присутствовать в частности следующие разделы:

- **const** – описание констант,
- **type** – описание типов данных,
- **var** – описание переменных,
- описание процедур и функций.

Имена функций и процедур являются их идентификаторами, именно по этим именам и выполняется вызов подпрограмм.

Список формальных параметров – это набор переменных, задаваемых при описании подпрограммы и используемых в ее теле в качестве средства получения исходных данных (*входные параметры*) и передачи результатов работы (*выходные параметры*).

Синтаксически каждый параметр оформляется следующим образом:

```
имя_параметра : тип
```

Несколько параметров в списке записываются через точку с запятой. В том случае, если необходимо объявить несколько параметров одного типа, их можно перечислить через запятую следующим образом:

имя_параметра1, имя_параметра2, имя_параметра3 : тип

Результатом функции является одно значение, тип которого совпадает с типом функции или неявно приводится к нему. Процедуры могут использоваться для выполнения некоторой последовательности действий, но также могут и в качестве результата своего выполнения возвращать одно или несколько значений. Оно (или они) передаются в основную программу как значения ее параметров. При вызове подпрограммы (процедуры или функции) ее формальные параметры заменяются фактическими в порядке их следования.

Фактические параметры – это параметры, которые передаются подпрограмме при обращении к ней.

Формальные параметры – это переменные, описанные в процедуре и определяющие тип и место подстановки фактических параметров. В теле подпрограммы действия производятся над формальными параметрами.

Число и тип формальных и фактических параметров должны совпадать с точностью до их следования.

Все формальные параметры делятся на два вида: *параметры-переменные* и *параметры-значения*.

Тело подпрограммы начинается с `begin`, а завершается `end` с последующей точкой с запятой. Даже если подпрограмма не содержит ни одного оператора, операторные скобки `begin` – `end` все равно должны быть записаны.

2. Глобальные и локальные переменные

В программе все переменные делятся на *глобальные* и *локальные*. *Глобальные переменные* – это те переменные, которые объявлены в описании основной части. *Локальные переменные* объявляются в подпрограммах. Под них выделяется память в стеке¹ при вызове подпрограммы, а при возвращении в точку вызова память освобождается. Таким образом, локальные переменные появляются при вызове подпрограммы, существуют только тогда, когда работает подпрограмма, и исчезают при завершении ее работы.

Существует система правил, определяющая *области видимости* (*действия, доступности*) переменных (аналогичные правила подходят и для констант, типов, процедур, функций и т. п.):

- переменная, объявленная в некоторой подпрограмме, является *локальной* для нее. Она «не видна» во всех других подпрограммах, за исключением вложенных, т. е. тех, которые описаны как внутренние для данной подпрограммы;
- если имя глобальной переменной используется для описания другой переменной во внутренней подпрограмме, то внутри подпрограммы работа будет вестись только с локальной переменной, причем в это время значение глобальной переменной будет недоступным. После завершения работы подпрограммы локальная переменная из стека удаляется, а глобальная переменная вновь начинает «работать», причем ее значение локальной переменной не изменяется.

Итак, переменная существует (т. е. размещена в оперативной памяти) только в то время, когда является активной подпрограмма, где она объявлена. Подпрограмма считается активной после начала ее выполнения и до завершения ее работы. Другими словами, память под переменные отводится только после входа в подпрограмму, где они описаны, и отбирается у них после завершения работы подпрограммы. Переменные, объявленные в основной программе, существуют все время работы программы (однако они не могут использоваться в подпрограмме, если там описана переменная с тем же именем). Таким образом, если значение какой-то переменной используется в нескольких подпрограммах, ее лучше описать как глобальную.

¹ Стек – специальная область оперативной памяти, которая используется, например, для хранения временных переменных (например, локальных переменных процедур), передачи параметров в процедуры.

Пример: ниже приведена программа, в которой в подпрограммах использованы и глобальные, и локальные переменные. В процедуре p1 использует локальная переменная i, имеющая тип char, а в процедуре p2 используется глобальная переменная i, имеющая тип integer.

```

program alphabet;

var i, j: integer;

procedure p1(k: integer);
var i: char;
begin
    i := chr(k);
    write(i);
end;

procedure p2;
begin
    i := i + 1;
end;

begin
    i := 97;
    for j := 1 to 26 do
        begin
            p1(i);
            p2;
        end;
    end.

```

В этой программе 26 раз выполняется следующее: процедура p1 выводит на экран символ, под номером, совпадающим со значением глобальной целочисленной переменной i (при этом внутри процедуры имя i соответствует символу, код которого равен k, а k – это формальный параметр, соответствующий фактическому параметру – глобальной переменной i), после чего в процедуре p2 значение глобальной переменной i увеличивается на единицу. В результате выполнения программы на экран выводятся строчные символы латинского алфавита от ‘a’ до ‘z’.

В общем случае можно сформулировать рекомендацию, состоящую в следующем: используйте обмен данными через глобальные переменные только тогда, когда без них нельзя обойтись.

Существует еще один механизм обмена данными между основной программой и подпрограммами – через параметры. В следующей главе различные способы обмена данными через параметры будут рассмотрены более подробно.

3. Передача параметров по значению и по ссылке

Локальные переменные хранятся в специальной области памяти, называемой *стеком*. Передача параметров функциям также осуществляется через стек. Однако передать можно как само значение переменной, так и её адрес в оперативной памяти. Отсюда и идет разделение на передачу параметров *по значению* и *по ссылке (адресу)*, а все формальные параметры соответственно делятся на два вида: *параметры-значения* и *параметры-переменные*. При этом следует помнить, что типы формального и фактического параметров обязательно должны совпадать.

3.1. Передача параметра по значению

При этом способе передачи параметров значения фактических параметров копируются в соответствующие области памяти формальных параметров, при этом фактические параметры изменить внутри подпрограммы нельзя, т. к. в действительности подпрограмма работает с копией переданных данных.

В качестве фактических параметров при передаче по значению можно использовать и переменные, и константы, и выражения. Такие параметры называются *параметры-значения*.

Внутри подпрограммы можно производить любые действия с данными формальными параметрами, допустимые для их типов, но любые изменения формальных параметров никак не отражаются на значениях соответствующих фактических параметров, т. е. какими они были до вызова подпрограммы, то такими же и останутся после завершения ее работы.

Рассмотрим пример, в котором происходит передача параметра по значению.

Пример. Составить программу, подсчитывающую число сочетаний без повторения из n элементов по k элементов. Число сочетаний без повторения вычисляется по формуле: $C_n^k = \frac{n!}{k!(n-k)!}$.

Обозначим через n и k переменные для хранения введенных чисел; c – переменная для хранения результата.

Чтобы подсчитать количество сочетаний без повторений необходимо вычислить $n!$, $(n-k)!$, $k!$.

Опишем функцию, вычисляющую факториал числа n ($n! = 1 \times 2 \times \dots \times n$).

```

function factorial(n: integer): int64;
var
    i: integer;
    res: int64;
begin
    res := 1;
    for i := 1 to n do res := res * i;
    factorial := res;
end;

```

Первая строка в описании функции – это ее заголовок. Служебное слово `function` (функция) указывает на то, что именем `factorial` названа функция. В скобках указан параметр `n` (число, факториал которого мы будем находить), который является целым числом (тип `integer`). Далее в заголовке указывается тип значения функции, ее результата. В данном примере результат функции `factorial` – целое число типа `int64` (мы взяли тип с большим диапазоном, поскольку значение функции факториал быстро возрастает и может превысить тип `integer`).

За заголовком функции следует описательная часть функции, которая в данном примере содержит только раздел переменных. Опишем переменные `i` (переменная для управления циклом) и `res` (для накопления значения факториала).

Далее идет раздел операторов (тело функции), в котором с помощью цикла подсчитывается значение факториала числа. Результат этого вычисления присваивается имени функции, таким образом, она и получает свое значение.

Напомним, что в теле функции можно использовать специальную переменную `result`, значение которой станет результатом функции. Нашу подпрограмму можно переписать следующим образом:

```

function factorial(n: integer): int64;
var i: integer;
begin
    result := 1;
    for i := 1 to n do result := result * i;
end;

```

Составим программу, использующую эту функцию, для вычисления количества сочетаний без повторений.

```

program combination;

var
    n, k: integer;
    a1, a2, a3, c: int64;

function factorial(n: integer): int64;
var i: integer;
begin
    result := 1;
    for i := 1 to n do result := result * i;
end;

begin
    writeln('Введите n, k для подсчета числа сочетаний:');
    readln(n, k);
    a1 := factorial(n);           {вычисление n!      }
    a2 := factorial(k);           {вычисление k!      }
    a3 := factorial(n - k);       {вычисление (n-k)! }
    c := a1 div (a2 * a3);       {результат          }
    writeln(c);
end.

```

При выполнении программы описание функции хранится в памяти компьютера. Действия функции выполняются тогда, когда в основной части программы необходимо найти значение факториала, то есть при обращении к этой функции. Ее вызов осуществляется в операторе присваивания. Обращение записывается в виде имени функции, за которым следует в круглых скобках параметр функции.

В нашей функции используется формальный параметр n . Тип фактических параметров определяется типом формального параметра – в нашем случае используется *int64*. Перед вычислением функции при каждом вызове формальным параметрам присваиваются значения фактических параметров: соответственно переменных n , k и выражения $(n - k)$.

Пусть $n=4$, а $k=2$. Когда в программе встретится оператор `a1 := factorial(n);` выполнятся следующие действия:

- выделится память для переменных, описанных в функции `factorial`;
- формальным параметрам присвоятся значения фактических параметров: формальный параметр n станет равен фактическому параметру n , т. е. четырем;

- выполнится программный код функции, то есть будет найден факториал числа 4;
- полученное значение функции передается в место вызова этой функции, то есть переменной `a1` присвоится значение 24, и осуществится переход к выполнению следующих действий основной программы.

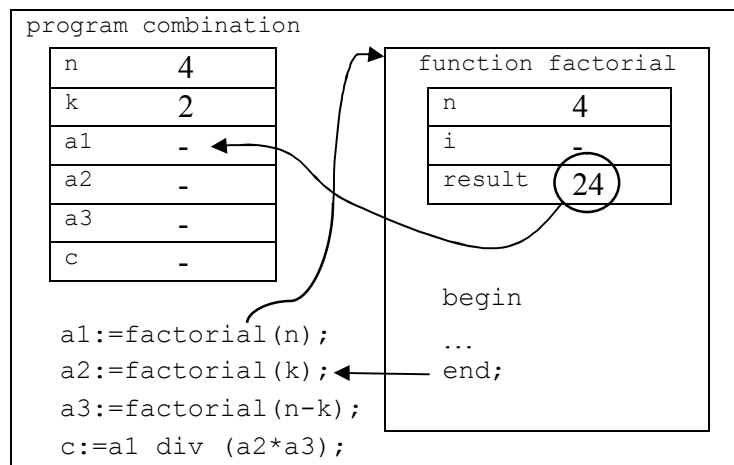


Рис. 2. Вызов подпрограммы

На рис. 2 показано распределение памяти между переменными, используемыми в функции и в основной программе. Прямоугольники соответствуют ячейкам памяти, стрелками показаны направления пересылок значений между ячейками. Обратите внимание на то обстоятельство, что переменной `n` (параметру в функции) отведена отдельная ячейка памяти, отличная от переменной `n` из основной части программы.

При выполнении оператора `a1 := factorial(n);` в ячейку `n` функции заносится $n = 4$. Вычисляется значение переменной `result` и передается (по последнему `end` функции) в точку вызова. При этом переменная `a1` получает значение, равное $n!$, то есть 24, а программа переходит к следующему оператору.

Операторы `a2 := factorial(k);` и `a3 := factorial(n - k);` еще дважды обращаются к функции `factorial`, передавая ей значения $k = 2$ и $n - k = 2$. Итак, всего в программе было 3 обращения к функции `factorial`, столько же раз были выполнены и описанные выше действия.

Еще раз подчеркнем, что описание функции – это самостоятельная часть программы, имеющая собственные переменные, которым отводится отдельное, не зависящее от основной программы место в памяти компьютера. Этим объясняется тот факт, что переменные, именуемые одним именем и

используемые как в описании функции, так и в основной программе, фактически являются разными переменными (в примере – переменная `n` основной части программы и параметр `n` в описании функции). При выполнении программы машина не путает имена этих переменных, так как области их действия не совпадают.

Таким образом, программист может вводить в описание функции различные имена, не заглядывая в другие части программы, что особенно важно при написании больших программ.

Пример. Рассмотрим программу, в которой используется функция с двумя параметрами-значениями. Формулировка задачи: с клавиатуры вводится два числа, необходимо найти наибольшее из них.

```
program maximum1;

var a, b: integer;

function get_max(x, y: integer): integer;
begin
    if x > y then result := x
    else result := y;
end;

begin
    write('Введите первое число ');
    readln(a);
    write('Введите второе число ');
    readln(b);
    writeln('Большее из двух чисел ', get_max(a, b));
end.
```

Заметьте, что здесь вызов функции происходит не в присваивании, а внутри операторы вывода.

Самостоятельно прорисуйте вызов функции `get_max` и передачу параметров (как на рис. 2).

3.2. Передача параметра по ссылке

При передаче параметра *по ссылке* любое изменение параметра внутри подпрограммы ведет за собой изменение соответствующего фактического параметра. Такие параметры называют *параметрами-переменными*. В качестве фактических параметров для них могут использоваться только переменные; константы или выражения использовать нельзя.

Параметры-переменные передаются в подпрограмму по ссылке, т. е. передается адрес фактического параметра. При изменении параметров-переменных в подпрограмме в действительности изменяются соответствующие им фактические параметры. Таким образом, если необходимо в точке вызова процедуры из программы получить после работы подпрограммы некоторые новые значения, соответствующие параметры нужно передавать по ссылке, т. е. оформлять их как параметры-переменные. Синтаксически передача параметра по ссылке отличается от передачи параметра по значению наличием ключевого слова `var` перед описанием формального параметра.

Пример. В следующей программе в процедуре используются два параметра-значения и один параметр-переменная. По результату выполнения эта программа не отличается от последнего примера.

```
program maximum2;  
  
var a, b, max: integer;  
  
procedure get_max(x, y: integer; var m: integer);  
begin  
    if x > y then m := x  
    else m := y;  
end;  
  
begin  
    write('Введите первое число ');  
    readln(a);  
    write('Введите второе число ');  
    readln(b);  
    get_max(a, b, max);  
    writeln('Большее из двух чисел ', max);  
end.
```

Пример. Составить программу для нахождения a^n , то есть n -й степени числа a , где a и n – целые числа ($n > 0$), вводимые с клавиатуры.

В решении будем использовать процедуру, в которой вычисляется степень некоторого числа.

```

procedure degree(x, y: integer; var res: int64);
var i: integer;
begin
    res := 1;
    for i := 1 to y do res := res * x;

end;

```

В процедуру передается три параметра: первый параметр – основание, то есть число, которое надо возвести в степень, второй параметр – это показатель, а третий – это результат. Первые два формальных параметра – это параметры-значения, а третий – это параметр-переменная и перед ним записывается слово `var`. Обозначим их x , y и res , где $res = x^y$. Все они описаны как целочисленные (x и y – тип `integer`, а res – `int64`, т. к. степенная функция быстро возрастает).

Вся программа для нашей задачи может иметь следующий вид:

```

program degree_finding;

var a, n: integer;
    s: int64;

procedure degree(x, y: integer; var res: int64);
var i: integer;
begin
    res := 1;
    for i := 1 to y do res := res * x;
end;

begin
    writeln('Введите два числа');    {ввод значений      }
    readln(a, n);
    degree(a, n, s);                 {вызов процедуры   }
    writeln('Результат ', s);        {вывод a^n        }
end.

```

Процедура вызывается как оператор, состоящий из имени процедуры. В круглых скобках передаются фактические параметры. В нашем примере,

фактические параметры a , n и s передают свои значения соответственно формальным параметрам x , y и res . После завершения работы процедуры переменные a и n имеют те же значения, что и при вызове, а s получает новое значение.

Пусть $a = 2$ и $n = 4$. Когда в программе встречается оператор `degree(a, n, s);` то выполняются следующие действия, проиллюстрированные на рис. 3:

- выделяется память для переменных, описанных в процедуре `degree`;
- формальным параметрам присваиваются значения фактических: x становится равным a , т. е. 2; y становится равным n , т. е. 4; res становится равным s ;
- выполняются операторы процедуры, то есть будет найдено значение 2^4 , т. е. 16;
- полученное значение присвоится переменной s , а переменные a и n остаются прежними, после этого происходит переход к выполнению следующих действий программы в точке вызова, то есть выполняется следующий оператор, стоящий за обращением процедуры.

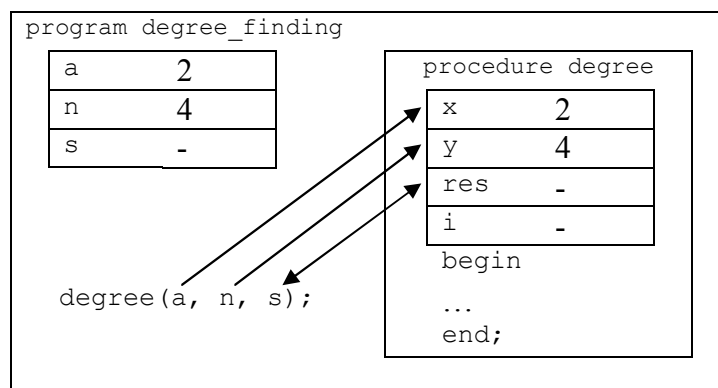


Рис. 3. Передача значений и ссылок на параметры

На рис. 3 показано распределение памяти между переменными, используемыми в процедуре и в основной программе. Прямоугольники соответствуют ячейкам памяти, каждая из них имеет свой адрес. Пересылка параметров идет по значению (\rightarrow) или по ссылке (\leftrightarrow), в первом случае передается только значение, а во втором – передается адрес ячейки, куда после выполнения процедуры будет записано новое значение. Для локальной переменной i и параметров x и y память выделяется только при обращении к данной процедуре, а после завершения ее работы соответствующая память освобождается. Обратите внимание, что под параметр res дополнительная

ячейка памяти не выделяется, поскольку процедуре передается адрес уже существующей переменной *s*.

После выполнения операторов (по последнему `end` в процедуре) идет возврат в точку вызова, при этом если фактические параметры передавались по значению, то они остаются неизменными, а если передавались по ссылке, то в ячейке с данным адресом появится новое значение (рис. 4):

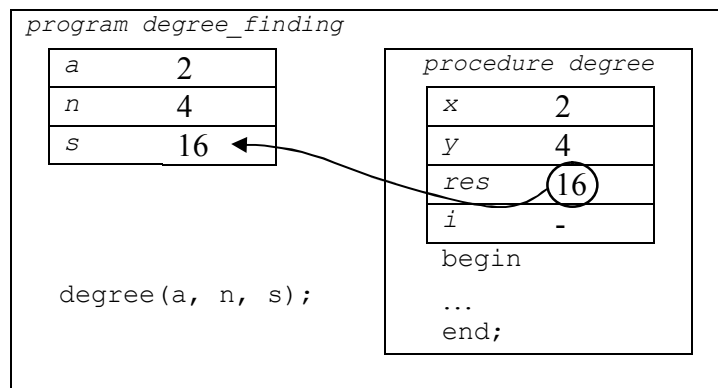


Рис. 4. Передача параметра по ссылке

Пример. Даны два целых числа. Поменять местами их значения.

Для перестановки напомним процедуру. Будем менять местами значения двух чисел, используя промежуточную переменную.

```

procedure swap(var x, y: integer);
var z: integer;
begin
  z := x; x := y; y := z;
end;
  
```

Здесь процедуре передаются два параметра, которые являются параметрами-переменными, то есть после выполнения они оба получат новые значения, значение переменной *a* будет равно исходному значению *b*, а значение *b* будет равно исходному значению *a*.

Самостоятельно напишите программу, в которой используется процедура `swap` и прорисуйте вызов этой процедуры, как на рис. 3 и 4.

Добавим еще несколько важных замечаний, касающихся передачи параметров по ссылке.

- В случаях, когда копирование данных из вызывающей программы в процедуру будет затратно по времени и/или по памяти, но при этом сами данные нельзя изменять, бывает необходимо передавать

фактический параметр по ссылке, но запретить его изменение. В таком случае перед описанием формального параметра ставится ключевое слово `const`.

- Считается неверным использовать функции, которые помимо своего значения имеют в качестве результата какой-либо параметр-переменную, хотя синтаксически это возможно.

3.3. Параметры по умолчанию

У программиста имеется возможность указать, какое значение будет принимать формальный параметр в том случае, если фактический параметр не был указан при вызове подпрограммы (процедуры или функции). Для этих целей используются *параметры по умолчанию*.

Например (функции с одним, двумя и тремя параметрами по умолчанию соответственно):

```
function sum(a, b: integer; c: integer := 0): integer;

function sum(a, b: integer; c: integer := 0;
             d: integer := 0): integer;

function sum(a, b: integer; c: integer := 0;
             d: integer := 0; e: integer := 0): integer;
```

Как видите, синтаксически это оформляется следующим образом: после описания формального параметра ставится оператор присваивания и указывается необходимое значение. При этом мы можем указать несколько параметров по умолчанию. Однако следует помнить, что все параметры по умолчанию указываются в конце списка формальных параметров. Кроме того, помните, что параметры по умолчанию должны всегда передаваться по значению.

Если при вызове подпрограммы не указать значение параметра по умолчанию, то будет использовано то значение, которое указано в описании подпрограммы.

Пример. Рассмотрим программу, в которой используется функция с двумя параметрами по умолчанию. В основной программе использовано 3 способа вызова функции `sum` – с двумя, тремя и четырьмя параметрами – слагаемыми.

```
program sum234;  
  
function sum(a, b: integer; c: integer := 0;  
              d: integer := 0): integer;  
  
begin  
    sum := a + b + c + d;  
end;  
  
begin  
    writeln(sum(1, 2));  
    writeln(sum(1, 2, 3));  
    writeln(sum(1, 2, 3, 4));  
end.
```

Поскольку есть несколько способов вызова функции `sum`, можно сказать, что эта функция является универсальной для сложения различного числа слагаемых (от двух до четырех).

4. Рекурсия

Рекурсивной подпрограммой называется подпрограмма, которая в процессе работы вызывает саму себя. Эту ситуацию можно сравнить с экраном телевизора, на котором показывают этот же телевизор, на экране второго – опять этот телевизор и так далее. Для того чтобы рекурсия не приводила к заикливанию, в теле рекурсивной подпрограммы всегда должен присутствовать условный оператор, одна из ветвей которого не содержит рекурсивных вызовов («заглушка рекурсии»).

При вызове рекурсивной процедуры в памяти размещаются параметры-значения, передаваемые процедуре, а также значения внутренних переменных вызываемой процедуры. Также запоминается адрес возврата в вызывающую процедуру, после чего управление передается вызванной процедуре. Суть заключается в том, что при каждом вызове создается новая копия со своими переменными, но как только она заканчивает свою работу, то память, занятая этими локальными переменными, освобождается, а полученные результаты передаются в точку вызова.

Типичная конструкция рекурсивной процедуры имеет вид:

```
procedure rec (t: integer);
begin
    действия на входе в рекурсию
    if условие then rec(t + 1);
    действия на выходе из рекурсии
end;
```

Проиллюстрируем работу рекурсивных подпрограмм на классическом примере вычисления факториала. Напомним, что факториал числа n равен произведению всех натуральных чисел от 1 до n :

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 2 \times 1 = n \times (n - 1)!$$

При такой записи $n!$ зависит от $(n - 1)!$, который, в свою очередь, зависит от $(n - 2)!$ и т. д. При этом $1! = 1$, что можно использовать как условие выхода из рекурсивной логики вычисления.

```
function factorial(n: integer): int64;
begin
    if n = 1 then factorial := 1
    else factorial := n * factorial(n - 1);
end;
```

Рассмотрим, как с помощью этой рекурсивной функции найти $4!$ (рис. 5). Первый вызов функции осуществляется из основной программы, например, `a := factorial(4);` – переменной `a` присваиваем значение $4!$. Действий на входе в рекурсию нет, этап вхождения выделен на рисунке жирными линиями. Он продолжается до тех пор, пока значение параметра не становится равным 1. После этого начинается выход из рекурсии (тонкие линии на рисунке). Таким образом, ход вычислительного процесса по управлению является нелинейным. Как отмечалось выше, в рекурсивной логике обязательно должно быть условие завершение процесса вхождения в рекурсию, называемое «заглушкой». Вход в рекурсию осуществляется вызовом процедуры или функции, а для реализации выхода из рекурсии требуется помнить точки возврата в вызывающую логику. Для этого в стеке сохраняются точки возврата, а также значения всех локальных переменных.

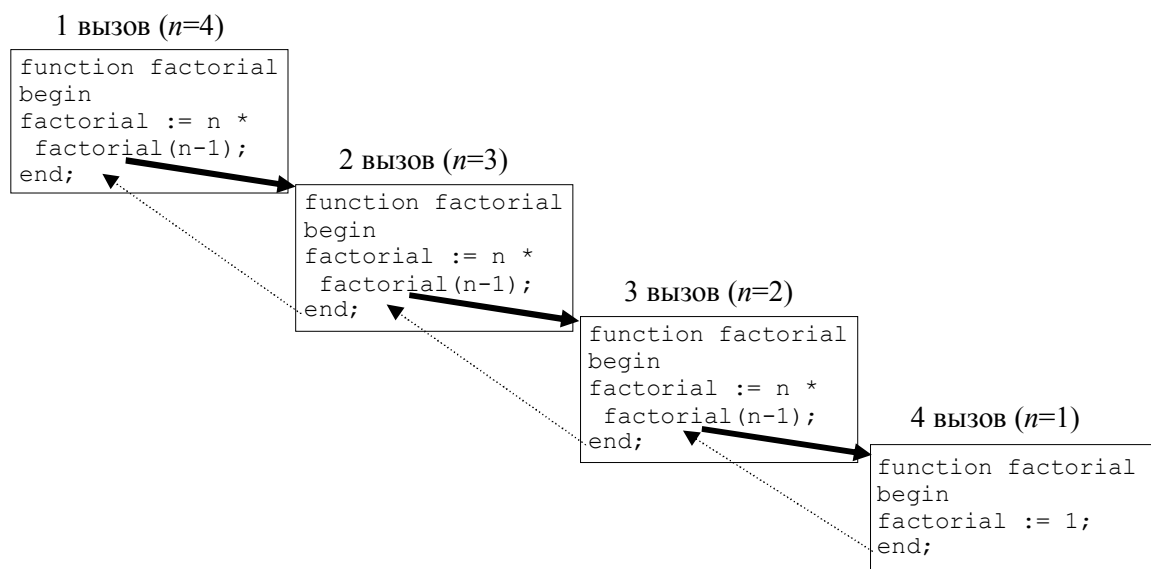


Рис. 5. Рекурсивные вызовы

Рекурсия сводит решение сложной задачи к ряду простых задач. В нашем примере для нахождения факториала в конечном счете выполняется просто ряд умножений, причем при каждом вызове функции не более одного умножения.

Рекурсивная подпрограмма может обращаться к себе прямо или косвенно (через другие процедуры или функции). Выше был приведен пример прямой рекурсии.

При более сложном виде рекурсии – *косвенной рекурсии* – все процедуры или функции в цепочке являются рекурсивными. В этом случае подпрограммы не пытаются вызвать непосредственно сами себя. Пусть у нас есть процедура А, которая вызывает процедуру В, а процедура В, в свою очередь, вызывает процедуру А. Т. к. в языке Pascal существует правило, согласно которому сначала необходимо объявить объект, а уже потом его использовать, непонятно, какую из процедур – А или В – нужно описывать первой. Предположим, что процедура В описана первой. Но она вызывает процедуру А. Тогда перед В необходимо вставить опережающее объявление, которое выглядит следующим образом:

```
procedure A(); forward;
```

После этого уже можно вставлять в код процедуру В:

```
procedure B();  
begin  
  ...  
  A();  
  ...  
end;
```

Далее вставляем код процедуры А, объявленной выше:

```
procedure A();  
begin  
  ...  
  B();  
  ...  
end;
```

В общем случае опережающее объявление выглядит следующим образом:

```
procedure имя (список_параметров); forward;  
function имя (список_параметров): возвращаемое_значение;  
  ...  
forward;
```

Поскольку косвенная рекурсия – достаточно сложный прием, советуем без особой необходимости его не применять.

В целом, рекурсию следует использовать обдуманно и только в тех случаях, когда без нее решить задачу сложнее, чем с ней, поскольку рекурсивные подпрограммы выполняются медленнее и требуют больше памяти, чем нерекурсивные (при каждом рекурсивном вызове сохраняются все локальные переменные вызывающей подпрограммы и выделяется память под локальные переменные для нового вызова). К преимуществам же рекурсии можно отнести то, что многие рекурсивные подпрограммы легко читаются; в некоторых алгоритмах изначально заложена идея рекурсии и с помощью этого приема они изящно реализуются.

Отметим, что любой рекурсивный алгоритм можно реализовать, не прибегая к рекурсии, хотя в некоторых случаях это сопряжено с большими усилиями.

Пример. Приведем пример рекурсивной функции, вычисляющей степень числа.

```
function degree(a: integer; x: integer): int64;
begin
  if (x = 1) then degree := a      {заглушка рекурсии }
  else degree := a * degree(a, x - 1);
                                {рекурсивный вызов}
end;
```

Самостоятельно прорисуйте вызовы рекурсии, если первый параметр равен 3, а второй параметр равен 5.

Эта функция возводит первый параметр в степень, равную второму параметру (он должен быть не меньше 1). Подпрограмму можно изменить, сделав вычисление степени числа нерекурсивным.

```
function degree(a: integer; x: integer): int64;
var res: int64;
    i: integer;
begin
  res := 1;
  for i := 1 to x do
    res := res * a;
  degree := res;
end;
```

5. Задания для самостоятельного выполнения

Реализовать следующие программы, используя процедуры и функции.

1. Даны координаты трех вершин треугольника. Найти длины всех его сторон.
2. Даны координаты трех вершин треугольника ABC и даны координаты четвертой точки D . Определить, является ли эта точка внутренней точкой треугольника.
3. Из первых N натуральных чисел вывести на экран те, сумма цифр которых нечетна.
4. Определить, чему равна максимальная первая цифра N введенных с клавиатуры чисел.
5. Найти числа из промежутка от A до B , у которых больше всего делителей.
6. Определить, является ли число совершенным, то есть равно ли оно сумме своих делителей, кроме самого себя.
7. Среди чисел из интервала от A до B найти все простые.
8. Составьте программу, проверяющую, является ли число палиндромом (например, число 12721 – палиндром).
9. Определить, является ли число автоморфным, то есть таким, квадрат которого заканчивается этим же числом (например, число 6, так как его квадрат 36 заканчивается на 6, или число 25 – его квадрат 625).
10. Используя функцию из предыдущей задачи, найти первые N автоморфных чисел, начиная с числа M .
11. Составить программу нахождения наибольшего общего делителя нескольких чисел, используя функцию нахождения НОД двух чисел.
12. Дано натуральное число N . Выяснить, имеются ли среди чисел N , $N + 1$, ..., $2N$ близнецы, т. е. простые числа, разность между которыми равна двум.

Следующие программы составить, реализовав рекурсивные подпрограммы.

1. Составить рекурсивную программу ввода с клавиатуры последовательности положительных чисел (окончание ввода – любое отрицательное число) и вывода ее на экран в обратном порядке.
2. Определите номер последнего члена арифметической прогрессии, не превышающего N .

3. Написать рекурсивную программу перевода числа из десятичной системы счисления в двоичную.

4. Найти первые N чисел Фибоначчи. Каждое число Фибоначчи равно сумме двух предыдущих чисел при условии, что первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21,...), поэтому в общем виде n -ое число можно определить так:

$$\Phi(n) = \begin{cases} 1, & \text{если } n=1 \text{ или } n=2 \\ \Phi(n-1) + \Phi(n-2), & \text{если } n > 2. \end{cases}$$

5. Используя следующее рекуррентное соотношение, вычислите сумму двух положительных целых чисел:

$$a + b = \begin{cases} a, & \text{если } b = 0 \\ (a+1) + (b-1), & \text{если } b \neq 0 \end{cases}$$

6. Написать рекурсивную программу вывода цифр следующим образом:

```
000000...000000 (N цифр 0)
11111...11111   (N-2 цифры 1)
2222...2222     (N-4 цифры 2)
...
888...888
99...99
888...888
...
2222...2222     (N-4 цифры 2)
11111...11111   (N-2 цифры 1)
000000...000000 (N цифр 0)
```

7. Написать рекурсивную программу вывода алфавита следующим образом:

```
abcde...vwxyz
bcde...vwxy
cde...vwxx
...
cde...vwxx
bcde...vwxy
abcde...vwxyz
```

В самом среднем ряду должно выводиться N букв (N выбирается случайным образом из диапазона (0..26]).

8. Составить рекурсивную программу-функцию подсчета количества разбиений натурального числа n , то есть его представлений в виде суммы натуральных чисел. Пусть, например, $N = 5$. Тогда разбиениями N являются

его представления в виде: 5; 4+1; 3+2; 3+1+1; 2+2+1; 2+1+1+1; 1+1+1+1+1 – всего 7.

9. Написать программу нахождения наименьшего десятичного натурального числа $N > 1$, оканчивающегося цифрой q , такого, что если перенести эту цифру из конца в начало N , то полученное число окажется в q раз больше N .

Литература

1. Баженова И. Ю., Сухомлин В. А. Введение в программирование: учебное пособие. – М.: Бином. Лаборатория знаний, Интуит, 2007.
2. Вирт Н. Алгоритмы и структуры данных. – СПб.: Невский диалект, 2008.
3. Окулов С. М. Основы программирования. – 5-е изд., испр. – М.: БИНОМ. Лаборатория знаний, 2010.
4. Павловская Т. А. Паскаль. Программирование на языке высокого уровня. – СПб.: Питер, 2007.